Desenvolvimento de um *toolkit* em *Silverlight* para detecção de faces em tempo real

Marcelo Cabral Ghilardi, Vinicius Gadis Ribeiro\*, Jorge Rodolfo Zabadal, Cristiana Andrade Poffal

Centro Universitário Ritter dos Reis – Faculdade de Informática – R. Orfanotrófio, 555, Alto Teresópolis – CEP 90840-440 – Porto Alegre – RS – Brasil

\* autor para correspondência. E-mail: vinicius@uniritter.edu.br

## RESUMO

A demanda por segurança em todos os setores, as práticas forenses e a identificação de criminosos fazem com que a detecção e o reconhecimento de faces em vídeos e fotografias, reconhecimento de íris e digitais, tornem-se um estudo importante. Além disso, há cada vez mais necessidade de múltiplas formas de interação homem/máquina. Controlar dispositivos por estímulos corporais é uma necessidade e tendência. Atualmente, por exemplo, alguns computadores, notebooks, telefones e videogames permitem interação a partir de suas câmeras, não apenas pela detecção de face, mas movimento corporal e detecção de objetos. A maioria destes dispositivos, além de câmera, possui acesso à Internet, o que cria uma gama de possibilidades ainda maior. Estas tendências tecnológicas justificam o desenvolvimento de um *toolkit* para a detecção de faces em tempo real. A escolha do *framework Silverlight* para o desenvolvimento deste *toolkit* proporciona que as aplicações que o utilizarem possam ser executadas em um navegador de internet. Este *toolkit* pode ser aprimorado para outras áreas de detecção e reconhecimento, por exemplo, reconhecimento de faces, de íris, movimento corporal e monitoramento de ambientes. Como prova de conceito, foi desenvolvida uma aplicação-exemplo.

Palavras-chave: Tecnologia, detecção de faces, cor de pele, *Silverlight*, framework.

## ABSTRACT

The demand for security, forensic practices and the identification of criminals, makes the detection and recognition of faces in videos and photographs, iris recognition, become an important study. Moreover, there is increasing need for multiple forms of human-machine interaction. Control devices for body stimulus are a need and trend. Currently, for example, some computers, laptops, phones and video games allow interaction from their cameras, not only for face detection, but body movement and object detection. The majorities of these devices have Internet access, which creates an even greater range of possibilities. These technological trends have prompted me to develop a toolkit for detecting faces in real time. The choice of Silverlight framework for the development of this toolkit provides that applications that use can be implemented in a web browser. This toolkit can be enhanced to other areas, for example, face recognition, iris recognition, body movement and environmental monitoring. An application example was developed as a proof of concept.

Keywords: Technology, face detection, skin color, Silverlight, framework.

## 1. INTRODUÇÃO

Sistemas automáticos de reconhecimento de cartões de crédito, cartões de banco, carta de habilitação de motorista, são exemplos em que duas imagens são comparadas, uma imagem alvo, obtida no ato do reconhecimento, é comparada à imagem contida no cartão, ou a um banco de imagens, onde esta imagem alvo deve possuir uma imagem equivalente. Também podem ser utilizados em práticas forenses ou identificação de criminosos, em que uma imagem extraída de uma câmera de segurança ou fotos é pesquisada em bancos de dados estabelecidos pela polícia. Ou ainda o reconhecimento de pessoas em acervos de vídeos de trânsito, shoppings etc. Hoje, vários sistemas de autenticação são validados pelo reconhecimento de face, íris, digitais ou voz.

Além disso, há cada vez mais necessidade de múltiplas formas de interação homem/máquina. Controlar dispositivos por estímulos corporais é uma tendência. Os novos videogames, como o WII, Playstation e Xbox, possuem módulos de interação com os jogos a partir de movimentos corporais. Computadores com acessibilidade para deficientes podem ser comandados por movimentos corporais ou até por estímulos cerebrais. Realidade aumentada é outra tendência; basicamente, é a integração do mundo real com o mundo virtual em tempo real e em três dimensões (AZUMA, 1993), onde objetos em vídeos são detectados e sobrepostos por outros objetos, animações ou informações (LIN; FAN, 2001).

As possibilidades para a criação de novos aplicativos que necessitem detectar objetos, faces, pessoas, são grandes (COSTA et ali, 2005; JENG et ali, 1998; LOPES, 2005; MORAES, 2006; VEZHNEVETS, 2003). Assim, essas possibilidades estimularam o desenvolvimento deste projeto. O foco é a criação de um *toolkit* que auxilie no desenvolvimento de aplicações para Web para detecção de faces em tempo real. O método de detecção de faces empregado no desenvolvimento do *toolkit* é pelo espetro de cores correspondentes à pele humana (ALATTER; RAJALA, 1999; CAI; GOSHTASBY, 1999; FILIPE, 2008; FILHO, 2006; KIM et ali, 2000; LOW, 2001; SUNG; POGGIO, 1998). A cor da pele humana tem pequena variação na crominância, apesar de diferentes intensidades (DAI; NAKANO, 1996; VEZHNEVETS, 2003; YACHIDA, WU, 1995). Na seção que se segue, são apresentadas algumas técnicas de detecção de face, bem como suas vantagens e desvantagens.

Como o *toolkit* deve permitir que aplicações desenvolvidas com ele possam ser executadas em navegadores de internet e tenham acesso a webcam da máquina cliente, foi escrito empregando o *framework Silverlight*. Este *framework* é desenvolvido pela Microsoft e permite criar tanto aplicações Web quanto aplicações desktop (BORCK, 2010; SCHULTE, 2011). Outra ferramenta disponível e concorrente para o desenvolvimento deste tipo de aplicação é o Adobe Flash. Ambas as tecnologias possuem características parecidas, mas a pesquisa realizada por Borck(2010) relata a superioridade do *Silverlight* em relação ao Flash, levando em consideração que, no presente trabalho, optou-se pelo desenvolvimento com o *Silverlight*.

Para validar o *toolkit* foi desenvolvida uma aplicação-exemplo que apresenta todos os passos para a detecção de faces. Foram testados vários tons de pele e vários cenários até chegar a um intervalo de cores de pele que se aplicou à maioria das pessoas e ambientes testados. O *toolkit* possui parâmetros para configurar novos ambientes e cores de pele. Pode ser aprimorado para desenvolver novos métodos de detecção ou reconhecimento de faces, de íris, movimento corporal ou até monitoramento de ambientes.

#### 2. MATERIAIS E MÉTODOS: O Método Base

O método que foi usado como base neste trabalho foi desenvolvido a partir dos métodos de análise de textura da pele. As motivações para a escolha residem na alta velocidade de processamento, podendo ser usado em tempo real por webcams e pela baixa complexidade do algoritmo que facilitou o desenvolvimento para *Silverlight.*

Mas a detecção de face por este método não é trivial, pois sofre influência de fatores como diversidade de cores de pele, diferentes pontos de vista de visualização, a iluminação facial e a qualidade de imagem da webcam utilizada.

Tendo em vista estas dificuldades, mas visando ao processamento em tempo real e utilização de webcams, foi projetada a arquitetura que será detalhada na sessão seguinte.

#### 2.1 Utilização do espaço de cor YCbCr

Antes do processamento da imagem, é preciso levar em conta que com a utilização da tecnologia *Silverlight*, as imagens são recebidas em matrizes pixels, onde cada pixel tem 32 bits representando o alpha, vermelho, verde e azul, ou seja, no formato ARGB - RGBA (*Red Green Blue Alpha*), embora seja por vezes descrita como um espaço de cor, na verdade é o modelo de cor RGB, com uma informação adicional, o canal Alpha.

O espaço de cor RGB - espaço de cor utilizado pelo Monitor de Tubos de Raios Catódicos - tem algumas desvantagens importantes quando se trata de filtragem de faixas de determinada cor, isto é, se desejar filtrar por textura de pele, é necessário um espaço que seja pouco influenciado pela luminosidade. No caso do espaço RGB, implicaria maior volume de espaços de cores, impossibilitando definir uma simples limiarização de cores. Existem outros espaços de cores que não sofrem do mesmo problema: o espaço de cor HSV - o sistema de cores formadas pelas componentes *hue* (matiz), *saturation* (saturação) e *value* (valor) -, por exemplo, define a cor em seus três componentes, *Hue*, *Saturation* e *Value* (brilho), onde a cor real (*Hue*) é representada por um círculo de 0º a 360º e o brilho é a altura de um cilindro.

Mas esse espaço é muito utilizado em algoritmos de detecção de pele, pois o custo computacional para a conversão de RGB para HSV é alto e se constitui um complicador para um projeto de detecção de face em tempo real. O espaço de cores YCbCr - espaço de cor caracterizado por uma transformação não-linear do espaço RGB, bastante utilizada pela padronização do sinal de TV da Europa - também permite limitar o espaço correspondente à textura de pele e seu custo computacional é menor que do espaço HSV, este espaço armazena o brilho no componente Y e a cor nos componentes Cb (Azul) e Cr (Vermelho). A conversão de RGB para YCbCr pode ser feita com uma simples adição e operações de multiplicação.

O processamento da imagem para a detecção de face é composta por quatro fases. A primeira fase é filtrar a imagem original selecionando pixels com valores contidos em limiares de cor de pele pré-estabelecidos e gerar uma imagem binária a partir do resultado do filtro. Na segunda fase, utiliza-se a imagem gerada e filtram-se ruídos, ou seja, pixels improváveis de pertencerem a uma face por estarem muito afastados de outros pixels, novamente gerando uma imagem binária resultado. Na terceira fase, dilatam-se os pixels restantes, pois na segunda fase podem ter se perdido pixels importantes durante a filtragem de ruídos. Na quarta fase, gera-se um histograma quantitativo, vertical e horizontal, para que se possa, a partir dos valores, delimitar a área da face. O resultado deste processamento são coordenadas x1,y1,x2,y2 da provável área de face na imagem original.

#### 2.2 Algoritmo de filtragem pela textura de pele

Para o desenvolvimento deste algoritmo, primeiramente deve-se definir o espectro ou limiares de cor de pele - o qual foram definidos a partir de testes com várias imagens -, na implementação há, duas propriedades que definem os limiares padrões máximos e mínimos, que podem ser alterados se necessários.

SkinLowerThreshold = new YCbCr(0.10, -0.25, 0.05);

SkinUpperThreshold = new YCbCr(1.00, 0.05, 0.20);

No algoritmo, demonstrado abaixo, recebe-se a imagem original (01) e cria-se uma imagem resultado (vazia) com o mesmo tamanho da imagem original (02), percorrem-se todos os pixels da imagem original (03), como o espetro de pele está em YCbBr convertem-se os pixels de RGB para YCbCr (04), verifica-se se o pixel está contido nos limiares estabelecidos para pele (05), se positivo, atribui-se ao pixel de mesma localização, mas na imagem resultado, um valor que define que este pixel é de pele (06). Os pixels restantes da imagem resultado ficam vazios, desta forma obtém-se uma imagem binária em que os pixels marcados representam pixels que podem ser pele (07).

**01** public WriteableBitmap ToSkin(WriteableBitmap imagem) {

**02** WriteableBitmap result = new WriteableBitmap(imagem.PixelWidth, imagem.PixelHeight);

var p = imagem.Pixels;

var rp = result.Pixels;

**03** for (int i = 0; i < p.Length; i++) {

**04** var ycbcr = YCbCr.FromARGB(p[i]);

**05** if (ycbcr.Y >= SkinLowerThreshold.Y && ycbcr.Y <= SkinUpperThreshold.Y

&& ycbcr.Cb >= SkinLowerThreshold.Cb && ycbcr.Cb <= SkinUpperThreshold.Cb

&& ycbcr.Cr >= SkinLowerThreshold.Cr && ycbcr.Cr <= SkinUpperThreshold.Cr) {

**06** rp[i] = ColorSkin; } }

**07** return result; }

Como se pode ver na Figura 1, a imagem original é uma imagem colorida e, após o processamento deste algoritmo, a imagem resultado é uma imagem binária que representa pixels que podem ser pele.

|  |  |
| --- | --- |
| Imagem original | Imagem resultado |

Figura 1- Imagens do filtro de pele

#### 2.3 Algoritmo para redução de ruídos

Este algoritmo percorre todos os pixels da imagem resultado do algoritmo anterior, quando o pixel for de pele, pesquisa todos os pixels vizinhos deste pixel, em uma distância de até três pixels para cada direção, se encontrar algum pixel vizinho vazio, limpa o pixel percorrido para que ele deixe de ser de pele, desta forma eliminando ruídos na imagem.

Abaixo o código do algoritmo, primeiro recebe a imagem resultado do algoritmo anterior (01), e cria-se uma nova imagem resultado (vazia) com o mesmo tamanho da imagem original (02), na matriz PosPix são informados todos os pixels vizinhos que serão visitados (03), iniciando pelo próprio pixel posição [0,0], depois o pixel da direita [0,1], e assim por diante, até chegar ao pixel mais distante [-3,3]. Percorrem-se todos os pixels da imagem, linha a linha (04), e para cada pixel de pele, percorrem-se os pixels vizinhos (05), se não encontrar nenhum pixel vizinho vazio (06), marca-se na nova imagem resultado o pixel correspondente com sendo mesmo de pele (07), desta forma obtém-se uma nova imagem binária nos quais são eliminados os ruídos da imagem (08). Foi implementada uma função que verifica se o pixel vizinho está vazio ou é de pele (09).

**01** public WriteableBitmap SkinFiltro(WriteableBitmap imagem) {

var p = imagem.Pixels;

var w = imagem.PixelWidth;

var h = imagem.PixelHeight;

**02** var result = new WriteableBitmap(w, h);

var rp = result.Pixels;

int i = 0;

int qtdPosPix = 49;

**03** int[,] PosPix = { {+0,+0},{+1,+0},{+1,+1},{+0,+1},{-1,+1},{-1,+0},{-1,-1},

{+0,-1},{+1,-1},{+2,-1},{+2,+0},{+2,+1},{+2,+2},{+1,+2},

{+0,+2},{-1,+2},{-2,+2},{-2,+1},{-2,+0},{-2,-1},{-2,-2},

{-1,-2},{+0,-2},{+1,-2},{+2,-2},{+3,-2},{+3,-1},{+3,+0},

{+3,+1},{+3,+2},{+3,+3},{+2,+3},{+1,+3},{+0,+3},{-1,+3},

{-2,+3},{-3,+3},{-3,+2},{-3,+1},{-3,+0},{-3,-1},{-3,-2},

{-3,-3},{-2,-3},{-1,-3},{+0,-3},{+1,-3},{+2,-3},{+3,-3}};

**04** for (int y = 0; y < h; y++) {

for (int x = 0; x < w; x++, i++) {

int iPosPix = 0;

bool bContinue = true;

**05** while (bContinue && iPosPix < qtdPosPix) {

**06** if (PixelIsEmpty(x+PosPix[iPosPix, 0], y + PosPix[iPosPix, 1], p, w, h))

{ bContinue = false; break; }

iPosPix++; }

if (bContinue)

**07** rp[i] = ColorSkin;} } return result; }

**09** private bool PixelIsEmpty(int x, int y, int[] p, int w, int h) {

bool isEmpty = false;

if (x > 0 && x < w && y > 0 && y < h) {

isEmpty = (p[y \* w + x] == ARGB.Empty); }

return isEmpty; }

Como se pode verificar na Figura 2, a imagem original é uma imagem com ruídos e, após o processamento deste algoritmo, a imagem resultado é uma imagem sem ruídos.

|  |  |
| --- | --- |
| Imagem original | Imagem resultado |

Figura 2- Imagens do filtro de ruídos

#### 2.4 Algoritmo para dilatação de pixel

O processo anterior de eliminação de ruídos pode eliminar pixels importantes da própria face, tornando-se necessário um algoritmo de dilatação dos pixels restantes, desta forma melhorando eventuais espaços em branco dentro da face que podem ser causados pode brilho excessivo, cabelo, óculos ou outros fatores.

O algoritmo de dilatação, ao receber a imagem resultado do algoritmo de ruído, percorre todos os pixels, para cada pixel vazio verifica seus vizinhos em uma distância de até três pixels para cada direção, se encontrar algum pixel vizinho de pele, marca o pixel atual como sendo de pele, desta forma dilatando todos os pixels já existentes de pele.

A seguir, segue o código do algoritmo, primeiro recebe a imagem resultado do algoritmo anterior (01), e cria-se uma nova imagem resultado (vazia) com o mesmo tamanho da imagem original (02), na matriz PosPix são informadas as posições dos pixels vizinhos que serão visitados (03) (de uma forma diferentes da implementada no algoritmo anterior, mas com o mesmo resultado), verificando os vizinhos mais próximos até os mais longínquos em até três pixels de distância. Percorrem-se todos os pixels da imagem, linha a linha (04), e para cada pixel vazio, percorrem-se os pixels vizinhos (05), se encontrar um pixel vizinho de pele (06), marca na nova imagem resultado o pixel correspondente com também de pele (07), desta forma obtém-se uma nova imagem binária onde estão dilatados todos os pixels da imagem recebida (08). Foi implementada uma função que verifica se o pixel vizinho é de pele ou vazio (09).

**01** public WriteableBitmap SkinDilate(WriteableBitmap imagem) {

var p = imagem.Pixels;

var w = imagem.PixelWidth;

var h = imagem.PixelHeight;

**02** var result = new WriteableBitmap(w, h);

var rp = result.Pixels;

int i = 0;

**03** int[] PosPix = { 0, -1, +1, -2, +2, -3, +3 };

int ixPosPix;

int iyPosPix;

bool bContinue;

**04** for (int y = 0; y < h; y++) {

for (int x = 0; x < w; x++, i++) {

ixPosPix = 0;

iyPosPix = 0;

bContinue = true;

**05** while (bContinue && ixPosPix < PosPix.Length) {

while (bContinue && iyPosPix < PosPix.Length) {

**06** if (PixelIsNotEmpty(x-PosPix[ixPosPix],y+PosPix[iyPosPix], p, w, h))

**07** { rp[i] = ColorSkin; bContinue = false; continue; }

iyPosPix++; }

ixPosPix++; } } }

**08** return result; }

**09** private bool PixelIsNotEmpty(int x, int y, int[] p, int w, int h) {

bool isNotEmpty = false;

if (x > 0 && x < w && y > 0 && y < h) {

isNotEmpty = (p[y \* w + x] != ARGB.Empty); }

return isNotEmpty; }

Como mostrado na Figura 3, a imagem original é uma imagem sem ruídos, mas com algumas áreas abertas na face, após o processamento deste algoritmo, a imagem resultado possui menos áreas abertas na face.

|  |  |
| --- | --- |
| Imagem original | Imagem resultado |

Figura 3- Imagens do algoritmo de dilatação

#### 2.5 Cálculo da área e posição da face

Tendo a imagem resultado do algoritmo de dilatação, cria-se uma matriz com a quantidade de pixels de pele por linha e coluna, isto é, X e Y da imagem, com estes valores, a primeira e a ultima posição com valor da matriz X são respectivamente as posições X1 e X2, e mesmo na matriz Y, onde a primeira e a última posição com valor da matriz Y são respectivamente as posições Y1 e Y2.

Para ampliar o filtro, são desconsideradas posições com valores menores que LimiteMínimoX e LimiteMínimoY. Após testes com imagens de diferentes tamanhos, chegou-se ao seguinte cálculo para definir os limites mínimos de valores para X e Y.

e

Para implementar este cálculo, foram criadas duas funções: a primeira função - GetHistogram() -, cria duas matrizes de inteiros, a matriz histX para as linhas (01) e a matriz histY para as colunas (02) da imagem, onde serão armazenadas a quantidade de pixels de cada linha e a coluna respectivamente, também armazena a maior quantidade de pixels encontrada em cada uma das matrizes X e Y (03). Para preencher as matrizes, percorrem-se todos os pixels da imagem (04) e soma cada pixel de cor de pele encontrado (05), compara a quantidade de cada posição de cada matriz para encontrar a maior quantidade de pixels (06).

public void GetHistogram(WriteableBitmap imagem) {

var p = imagem.Pixels;

var w = imagem.PixelWidth;

var h = imagem.PixelHeight;

**01** histX = new int[w];

**02** histY = new int[h];

**03** histMaxX = 0; histMaxY = 0;

**04** for (int y = 0; y < h; y++) {

for (int x = 0; x < w; x++) {

**05** if (p[y \* w + x] == ColorSkin) {

histX[x]++; histY[y]++;

**06** if (histX[x] > histMaxX) { histMaxX = histX[x]; }

if (histY[y] > histMaxY) { histMaxY = histY[y]; } } } } }

Já na segunda função, GetLimits(), utiliza-se o resultado das matrizes da função anterior para calcular a posição da face, percorrem-se todas todos os itens de cada matriz, desconsiderando valores menores que os pré-definidos (01), para a matriz X toma-se a primeira posição com valor para marcar a posição X1 (02) e a última posição com valor para marcar a posição X2 (03), o mesmo para a matriz Y, toma-se a primeira posição com valor para marcar a posição Y1 (04) e a última posição com valor para marcar a posição Y2 (05).

public void GetLimits(WriteableBitmap imagem) {

var p = imagem.Pixels;

var w = imagem.PixelWidth;

var h = imagem.PixelHeight;

X1 = w - 1; Y1 = h - 1; X2 = 0; Y2 = 0;

**01** int limitMínimoX = w / 20;

int limitMínimoY = h / 20;

for (int x = 0; x < w; x++) {

if (histX[x] > limitMínimoX) {

**02** if (x < X1) { X1 = x; }

**03** if (x > X2) { X2 = x; } } }

for (int y = 0; y < h; y++) {

if (histY[y] > limitMínimoY) {

**04** if (y < Y1) { Y1 = y; }

**05** if (y > Y2) { Y2 = y; } } }

if (X1 == (w - 1) || Y1 == (h - 1)) {

X1 = 0;

Y1 = 0; } }

#### 2.6 Implementação do *toolkit*

O *toolkit* foi desenvolvido com o IDE (*Integrated Development Environment*) Visual Studio 2010, em um notebook equipado com webcam e sistema operacional Windows 7. O Visual Studio 2010 é um IDE da Microsoft que pode ser utilizada para desenvolver aplicações dos mais variados tipos e para diversas finalidades. O programa disponibiliza tanto  o uso de console de comandos para programação quanto da ferramenta gráfica. O aplicativo exemplo criado para demonstrar a utilização do *toolkit* foi desenvolvido no mesmo IDE do *toolkit*. A homologação e testes foram executados nos navegadores de internet Internet Explorer versão 8 e no Google Crome versão 10, no próprio notebook de desenvolvimento e também em um desktop equipado com uma webcam 2.0 e sistema operacional Windows 7.

O *toolkit* foi desenvolvido na plataforma .NET e escrito na linguagem de programação C#, escrito de forma que possa ser utilizado em aplicativos desenvolvidos no *framework Silverlight*.

C# é uma linguagem de programação orientada a objetos, fortemente tipada, desenvolvida pela Microsoft como parte da plataforma .NET. A sua sintaxe foi baseada no C++ e com influência da linguagem de programação Java.

O *framework Silverlight* faz parte da plataforma .NET e permite criar aplicações RIA (*Rich Internet Applications*), é uma alternativa para a ferramentas de desenvolvimento Adobe Flash e Adobe Flex que possuem características parecidas. A partir da versão 4.0 do *Silverlight* foi incorporada a possibilidade de acessar equipamentos da máquina cliente como webcam, impressora, microfone etc. Atualmente, o *Silverlight*, desenvolvido pela Microsoft, executa em ambiente Windows e Mac, e há uma implementação do *Silverlight*,de codinome Moonlight, financiado pela Microsoft e Novel, que executa no sistema operacional Linux.

RIA são aplicações Web, mas com características de aplicações Desktop, Microsoft *Silverlight*, Adobe Flash, HTML 5.0, são exemplos de ferramentas para o desenvolvimento de aplicações RIA.

.NET é uma plataforma para desenvolvimento e execução de sistemas e aplicações. Todo código gerado para .NET pode ser executado em qualquer dispositivo que possua um *framework* .NET. Atualmente, permite-se desenvolvimento para os sistemas operacionais Windows, e há implementações paralelas para sistemas operacionais com Linux e MacOs.

#### 2.6.1 Classes do *toolkit*

O *toolkit* foi desenvolvido com quatro classes; dessas, três são ferramentas para os espaços de cor - as classes ARGB, RGB e YCbCr e uma classe para a detecção de face, classe FaceDetect. A classe ARGB possui constantes de cores em formato ARGB, é necessária para evitar o custo do cálculo de conversão do formato RGB para ARGB.

A classe RGB armazena uma cor no formato RGB, possui métodos de conversão para o formato ARGB. A classe YCbCr armazena uma cor no formato YCbCr, possui métodos de conversão para o formato RGB.

A classe FaceDetect possui os algoritmos de detecção de faces implementados para este trabalho e demonstrados no início deste capítulo. Há alguns métodos extras, como ToGray e ToLine, que são métodos de estudo no desenvolvimento do trabalho e que foram mantidos na classe, para futuras implementações.

#### 3. RESULTADOS E DISCUSSÃO

A aplicação possui uma tela que demonstra todos os passos da detecção de faces implementada neste trabalho, Figura 4. A seguir, a definição de cada área da figura.

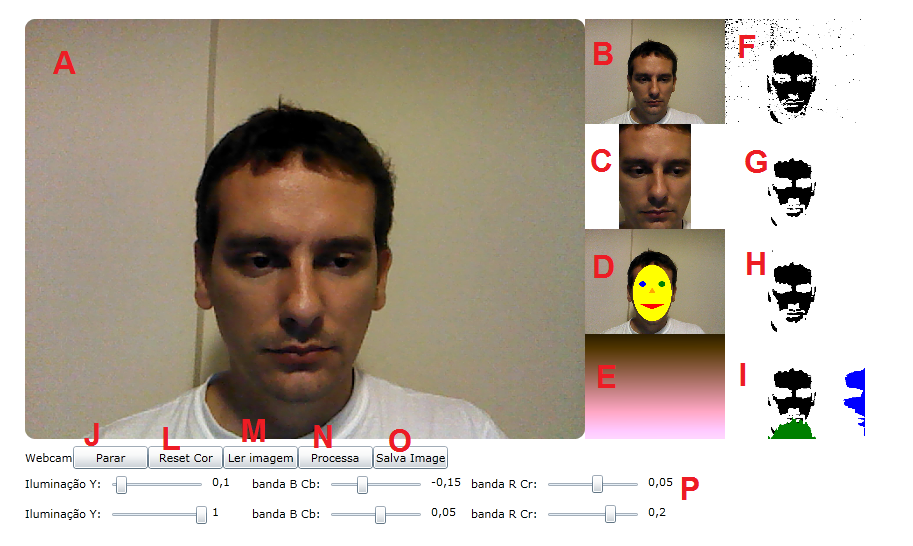


Figura 4 - Tela da Aplicação Exemplo

* A área “A” é igual a área “B”, mas a área “A” pode receber qualquer imagem das áreas “B” até “I”, para isso o usuário deve clicar na área desejada.
* A área “B” possui a imagem recebida da webcam ou liga do computador do usuário.
* A área “C” possui apenas a face detectada na imagem.
* A área “D” possui a imagem principal com um SMILE sobre a face detectada.
* A área “E” é uma representação do espetro de cor de pele definido.
* A área “F” é a imagem binária após o filtro de pele.
* A área “G” é a imagem binária após o filtro de ruídos.
* A área “H” é a imagem binária após a dilatação dos pixels restantes.
* A área “I” é uma representação gráfica do histograma.
* A área “J”, botão para iniciar ou parar a webcam.
* A área “L”, botão para re-estabelecer as configurações padrões da área “P” sobre o espetro de cor.
* A área “M”, botão para ler uma imagem do computador do usuário.
* A área “N”, botão para processar a imagem lida do computador.
* A área “O”, botão para salvar a imagem da área “D”, e acrescenta na imagem um texto com as informações de configuração usada no espetro de cor.
* A área “P”, configurações do espetro de cor.

#### 3.1 Bibliotecas empregadas: ImageTools e WriteableBitMap

Na aplicação exemplo, foi implementada a funcionalidade de acesso e gravação de imagens PNG e JPG na máquina cliente, empregando a biblioteca da ImageTools – disponível no site: http://imagetools.codeplex.com. Para simplificar o acesso às imagens, foi utilizada a biblioteca ‘ImageTools for Silverlight’. É uma biblioteca de código aberto que fornece funcionalidades carregamento e manipulação de imagens em diferentes formatos.

Tanto no *toolkit* quanto na aplicação, manipulam-se imagens no formato WriteableBitmap. A classe original fornecida no *Silverlight* não possui todos os métodos necessários para a implementação deste trabalho, possui apenas funções simples de acesso aos pixels da imagem, desta forma foi necessário utilizar a biblioteca de código aberto WriteableBitmapEx que é uma coleção de métodos de extensão para Silverlight WriteableBitmap, pode ser encontrada em http://writeablebitmapex.codeplex.com/.

#### 3.2 Obtendo o espetro de cor de pele

Para obter o valor mínimo e máximo do espaço de cor YCbCr, foi utilizado um conjunto de 13 imagens, pessoas com cor de pele diferente, caucasianos, amarelos, negros e em diferentes ambientes ou complexos. O trabalho foi conduzido manualmente e individualmente para cada imagem. Após testes em todas as imagens submetidas, obteve-se como resultado o Quadro 1, disponível no apêndice.

Deve-se destacar que, a partir da imagem de número 06, em algumas imagens os parâmetros de cor não se encaixam nos valores pré-determinados – alterações no padrão destacadas em vermelho. Porém, como foi apresentada, com alguns ajustes nos parâmetros, a detecção se torna eficiente.

#### 3.3 Análise de imagens em situações adversas: fundo complexo, face pequena e iluminação diversa

Nas Figuras 5, 6 e 7, ao processar a imagem com os valores pré-definidos, o ambiente complexo interferiu no filtro de ruídos, o resultado são grandes áreas que não são faces ainda presentes na imagem binária.



Figura 5- Imagem com fundo complexo.



Figura 6- Imagem com fundo complexo.

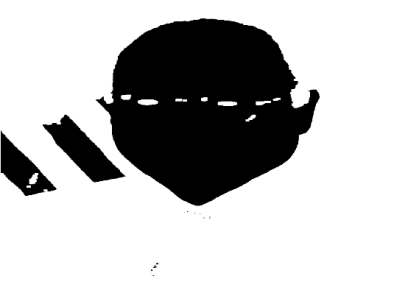


Figura 7- Imagem com fundo complexo.

Já nas imagens 8, 9 e 10, as faces são muito pequenas em comparação ao todo da imagem e o ambiente é complexo, os algoritmos implementados não são apropriados para este tipo de imagem.



Figura 8- Imagem com face pequena e fundo complexo.



Figura 9- Imagem com face pequena e fundo complexo.



Figura 10- Imagem com face pequena e fundo complexo

Na Figura 11, a face possui boa iluminação frontal, e pouca iluminação no ambiente, pela baixa iluminação do ambiente, o ambiente não é considerado complexo, tornando-se fácil a detecção da face.

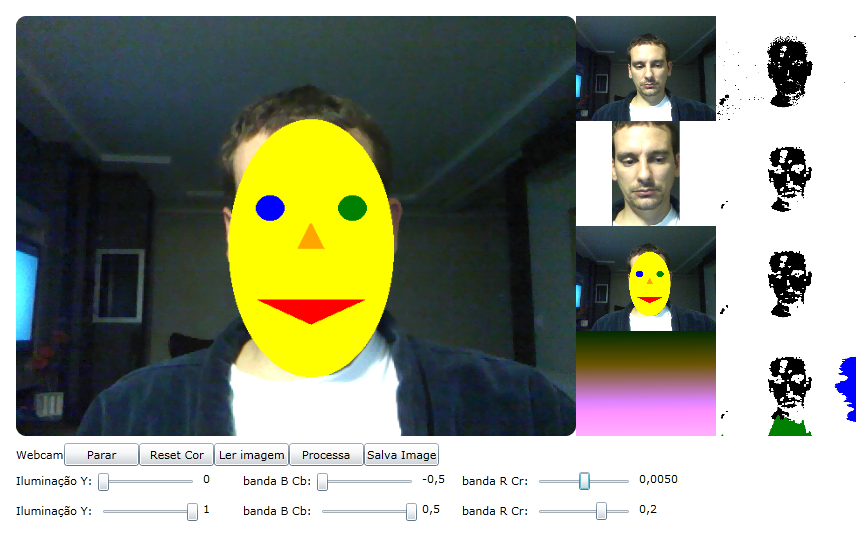


Figura 11- Imagem com iluminação frontal

Na Figura 12, a face possui boa iluminação frontal, mas o ambiente também possui boa iluminação, desta forma salientam as cores de fundo, deixando o ambiente um pouco mais complexo.

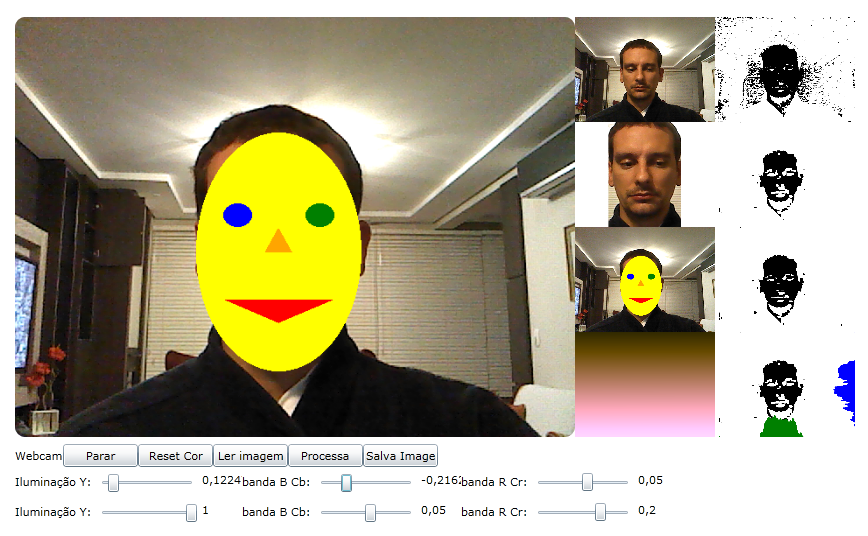


Figura 12- Imagem com iluminação total

Na Figura 13, a face possui pouca iluminação frontal, mas o ambiente possui boa iluminação, desta forma a face se torna apenas uma sombra, mas com ajustes nos parâmetros foi possível detectar a face.

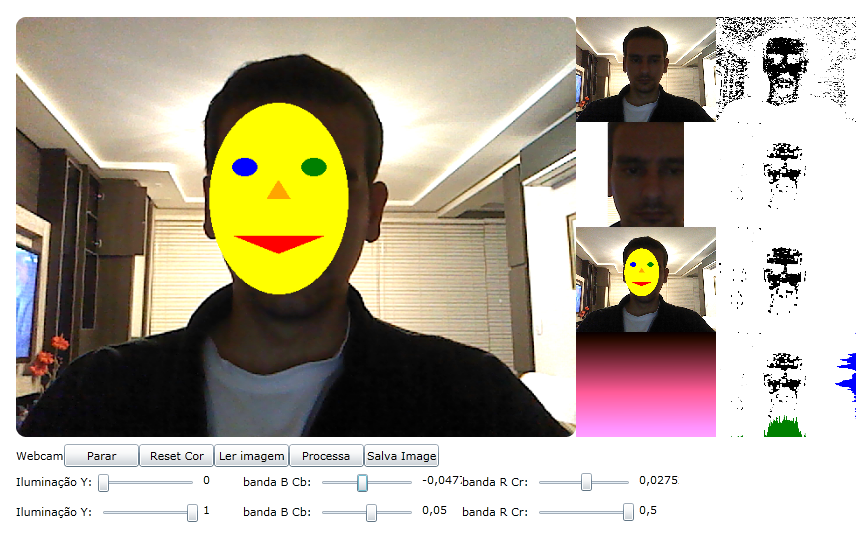


Figura 13- Imagem sem iluminação frontal

Na Figura 14, a face e o ambiente quase não possuem iluminação, mas há um pequeno foco de luz no lado esquerdo da imagem; este foco de luz foi o suficiente para a detecção da face.

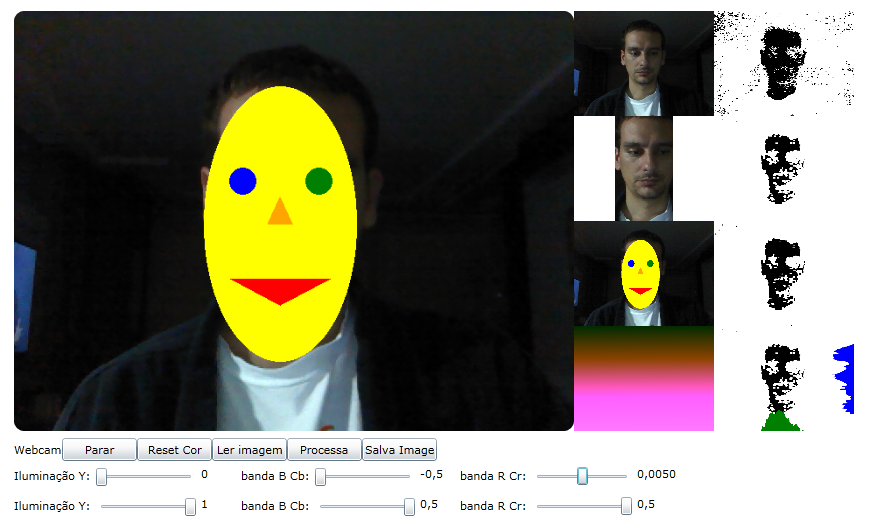


Figura 14- Imagem com pouca iluminação lateral

## 4. CONCLUSÕES

O desenvolvimento do *toolkit* para detecção de faces foi desenvolvido como proposto e, para demonstrar sua funcionalidade, foi implementado um programa exemplo que explora toda sua funcionalidade e desempenho.

Com base nos resultados obtidos do presente trabalho, e da pesquisa realizada para o mesmo, é possível efetuar algumas considerações. Mesmo utilizando uma pequena base de imagens, o algoritmo foi eficiente para pessoas com diferentes cores de pele, de cor branca a cor negra. A principal dificuldade do algoritmo são os ambientes complexos. O algoritmo não foi implementado para encontrar mais de uma face por imagem. Nas máquinas testadas é quase imperceptível o tempo de processamento da imagem, desta forma podendo ser classificado como um sistema em tempo real.

Para continuidade deste “toolkit”, acrescentar algoritmos para separar as áreas de possíveis faces da imagem e para validação de face, eliminaria o problema de ambientes complexos e o problema de mais de uma face na mesma imagem.

A partir deste trabalho é possível implementar algoritmos de reconhecimento de face, tanto para imagens como para vídeos, podendo ser usado em logins de sites, pesquisa de pessoas em bibliotecas de imagens ou ainda reconhecimentos de pessoas em vídeo gravado ou em tempo real.

## REFERÊNCIAS

AZUMA, Ronald. Tracking Requirements for Augmented Reality. **Communications of the ACM**, 36, 7 (July 1993), 50-51.

ALATTER, A. e RAJALA, S. **Facial Features Localization in Front View Head And Shoulders Images**, IEEE International Conference on Acoustics, Speech and Signal Processing, 1999, pp.3557-3560.

BORCK, James R. **Microsoft Silverlight 4 vs. Adobe Flash 10.1**. InfoWorld. 18/10/2010. Disponível em: http://www.infoworld.com/d/developer-world/infoworld-review-microsoft-silverlight-4-vs-adobe-flash-101-260

COSTA, Alex da; RODRÍGUEZ , Antonio Gabriel; SIMAS, Etiene P. Lazzeris; Araújo, Roberto da Silva. **Lógica fuzzy: Conceitos e aplicações**. Relatório. Universidade do Vale do Rio dos Sinos – UNISINOS – 2005

CAI, J. e GOSHTASBY, A.A. Detecting human faces in color images. **Image Vision Comput.**, 18(1):63–75, 1999.

DAI, Y. e NAKANO,Y. Face-texture model based on SGLD and its application in face detection in a color scene. **Pattern Recognition**, 29(6):1007–1017, 1996.

FILIPE, Silvio Brás. **Detecção de faces humanas em tempo real**. 2008. 67f. Dissertação (Licenciatura em Engenharia Informática). Covilhã, Portugal: Universidade da Beira Interior, 2008.

FILHO, Heitor Soares Ramos. **Detecção de Pele Humana em Imagens Detectadas**

**na Web**. Dissertação de Mestrado. Maceió: Universidade Federal de Alagoas, 2006.

JENG, Shi-Hong, Hong Y. Liao, Chin C. Han, Ming Y. Chern, and Yao T. Liu. Facial feature detection using geometrical face model: An ancient approach. **Pattern Recognition**, 31(3):273–282, March 1998.

KIM, H.S., KIM, E.Y., HWANG, S.W. e KIM, H.J. **Object-based human face detection**. Consumer Electronics, 2000. ICCE. International Conference on Consumer Electronics, pages 354–355, 2000.

LIN, C. e FAN, K.C. Triangle-based approach to the detection of human face. **Pattern Recognition**, 34(6):1271–1284, 2001.

LOPES, Eduardo Costa. **Detecção de Faces e Características Faciais**. Porto Alegre: PUCRS, Pós-Graduação em Ciência da Computação, 2005.

LOW, E.H. e KEE, B. Face detection: A survey. **Computer Vision and Image Understanding**, 83(3):236–274, September 2001.

MORAES, Denis Altieri de Oliveira. **Algoritmo para Suavizamento de Imagens Digitais por Filtragem Gaussiana em Connection Machines**. Porto Alegre: UFRGS, Programa de Pós-graduação em Computação, 2006.

SCHULTE, Rene. FaceLight – **Silverlight 4**: Real-Time face detection - http://rene-schulte.info/. 09/03/2011.

SUNG, K.K. e POGGIO, T. **Example-based learning for viewbased human face detection**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(1):39–51, 1998.

VEZHNEVETS, V. **Face and Facial Features Tracking for Natural Human-Computer Interface.**  Disponível na Web por http em: http://www.graphicon.ru/2002/pdf/Vezhnevets\_En\_Re.pdf . Acesso em novembro de 2010.

WANG, J. e TAN, T. A new face detection method based on shape information. **Pattern Recogn. Lett.**, 21(6-7):463–471, 2000.

YACHIDA, M. e WU, Q.C.H. **Face detection by *fuzzy* pattern matching**. In ICCV ’95: Proceedings of the Fifth International Conference on Computer Vision, page 591, Washington, DC, USA, 1995. IEEE Computer Society.

**APÊNDICE** – Resultados obtidos

Este quadro mostra que a maioria das faces é detectada no seguinte espectro de cor:

Dados:   
 e

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | FOTO | DETECÇÃO | CONFIGURAÇÃO | | | | | |
| Y | | Cb | | Cr | | |
| Min | Max | Min | Max | Min | Max | |
| 1 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto1.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n1.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 2 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto2.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n2.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 3 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto3.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n3.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 4 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto4.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n4.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 5 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto5.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n5.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 6 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto6.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n6.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,09 | 0,2 | |
| 7 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto7.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n7.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 8 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto8.jpg |  | 0,1 | 1 | -0,25 | 0,05 | 0,05 | 0,2 | |
| 9 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto9.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n9.jpeg | 0,1 | 1 | -0,25 | 0,05 | 0,09 | 0,26 | |
| 10 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto10.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n10.jpeg | 0,15 | 1 | -0,25 | 0,05 | 0,02 | 0,18 | |
| 11 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto11.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n11.jpeg | 0,35 | 0,75 | -0,25 | 0,05 | 0,16 | 0,18 | |
| 12 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto12.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n12.jpeg | 0,1 | 0,62 | -0,25 | 0,05 | 0,05 | 0,18 | |
| 13 | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\foto13.jpg | Descrição: Descrição: C:\Users\Marcelo Cabral\Dropbox\Trabalho Final\Images\xxx\n13.jpeg | 0,41 | 1 | -0,25 | 0,05 | 0,05 | 0,18 | |

Quadro 1 – Resultados obtidos